

A proposal to a generalised splicing with a self assembly approach

L Jeganathan, R Rama, and Ritabrata Sengupta

Department of Mathematics
Indian Institute of Technology
Chennai 600 036, India
lj, ramar, rits@iitm.ac.in

Abstract. Theory of splicing is an abstract model of the recombinant behaviour of DNAs. In a splicing system, two strings to be spliced are taken from the same set and the splicing rule is from another set. Here we propose a generalised splicing (GS) model with three components, two strings from two languages and a splicing rule from third component. We propose a generalised self assembly (GSA) of strings. Two strings u_1xv_1 and u_2xv_2 self assemble over x and generate u_1xv_2 and u_2xv_1 . We study the relationship between GS and GSA. We study some classes of generalised splicing languages with the help of generalised self assembly.

1 Introduction

Tom Head proposed [5] an operation called ‘splicing’, for describing the recombination of DNA sequences under the application of restriction enzymes and ligases. Given two strings $u\alpha\beta v$ and $u'\alpha'\beta'v'$ over some alphabet V and a splicing rule $\alpha\#\beta\$\alpha'\#\beta'$, two strings $u\alpha\beta'v'$ and $u'\alpha'\beta v$ are produced. The splicing rule $\alpha\#\beta\$\alpha'\#\beta'$ means that the first string is cut between α and β and the second string is cut between α' and β' , and the fragments recombine crosswise.

The splicing scheme (also written as H-scheme) is a pair $\sigma = (V, R)$ where V is an alphabet and $R \subseteq V^*\#V^*\$V^*\#V^*$ is the set of splicing rules. Starting from a language, we generate a new language by the iterated application of splicing rules in R . Here R can be infinite. Thus R can be considered as a language over $V \cup \{\#, \$\}$. Splicing language (language generated by splicing) depends upon the class of the language (in the Chomskian hierarchy) to be spliced and the type of the splicing rules to be applied. The class of splicing language $H(FL_1, FL_2)$ is the set of strings generated by taking any two strings from FL_1 and splicing them by the strings of FL_2 . FL_1 and FL_2 can be any class of languages in the Chomskian hierarchy. Detailed investigations on computational power of splicing is found in [9].

Theory of splicing is an abstract model of the recombinant behaviour of the DNAs. In a splicing system, the two strings to be spliced are taken from the same set and the splicing rule is from another set. The reason for taking two strings from the same set is, in the DNA recombination, both the objects to be spliced are DNAs. For example, the splicing language in the class $H(FIN, REG)$ is the language generated by taking two strings from a finite language and using strings from a regular language as the splicing rules. Any general ‘cut’ and ‘connection’ model should include the cutting of two strings taken from two different languages. The strings spliced and the splicing rules have an effect on the language generated by the splicing process. In short, we view a splicing model as having three components, two strings from two languages as the first two components, and a splicing rule as the third component. Our proposal of a generalised splicing model (a formal definition of GS: Generalised splicing, is given in section 2 definition 1) will be:

$$GS(L_1, L_2, L_3) := \{z_1, z_2 : (x, y) \models_r (z_1, z_2), x \in L_1, y \in L_2, r \in L_3\}.$$

Instead of taking two strings from same language, as being done in the theory of splicing, we take them from two different languages. We cut them by using rules from a third language. This means, taking an arbitrary word $w_1(\in L_1)$ and an arbitrary word from $w_2(\in L_2)$, we cut them by using an arbitrary rule of L_3 . If $L_1 = L_2$ in the generalised splicing model, we get the usual H -system.

The motivation of the above proposal of a generalised theory of splicing comes from the self assembly of strings [4]. Two strings uv and vw self assemble over v and generate uvw . Here, the overlapping strings appear at the end of one string and at the beginning of the other. Then comes the question: What will be the generalisation if we do not restrict the overlapping strings to be in the end (or the beginning) of the strings that participate in the assembling process. As an answer to the above question, we propose a generalised self assembly (GSA) of two strings (definition 2). Two strings u_1xv_1 and u_2xv_2 self assemble over the sub-string x and generate the strings u_1xv_2 and u_2xv_1 , as illustrated in the right hand side of the figure 1. The generated words indicate that

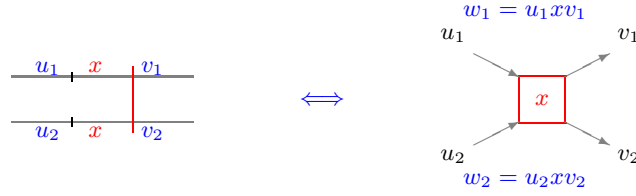


Fig. 1. Equivalence of generalised splicing and self assembly

the x -self assembly of w_1 and w_2 (self assembly with x as the overlapping string) is just a generalised splicing of w_1 and w_2 with a splicing rule $x\#\$x\#$.

We take advantage of this equivalence of GS and GSA and plan to investigate the generalised splicing for some classes of languages in Chomskian hierarchy. Since an investigation of the classes of languages under the generalised splicing model is going to be a more complicated one, compared to the existing H -system in all sense, we narrow down the investigation of the generalised splicing model by taking $L_3 = V^+ \cup \{(w_1, w_2) : w_1 \in L_1, w_2 \in L_2\}$ (V is the set of common symbols that appear in L_1 and L_2), which constitutes the set of splicing rules: a word $w \in V^+$ indicates that the splicing rule will be $w\#\$w\#$, where $\#$ and $\$$ have the usual meanings as in H -system; a pair of words $(w_1, w_2) \in L_3$ indicate that the splicing rule will be of the form $w_1\#\$w_2\#$. The very purpose of including the pair (w_1, w_2) in L_3 is to include the words that are being spliced, in the set of words generated by the GS. The necessity of including the parent words is discussed at the end of section 2.

Though the whole theory of splicing can be rewritten with the generalised splicing system, nevertheless, in this paper, we investigate $GS(L_1, L_2, L_3)$ for $L_1, L_2 \in \{REG, LIN, CF\}$ and L_3 is as given in the previous paragraph. For an investigation, we define the GSA of automata, regular grammar, linear grammar, context free grammar (apart from the GSA of two languages). In this paper, section 2 discusses the definitions of GS and GSA. The subsequent sections discuss the generalised self assembly of finite languages, regular languages, linear languages and the context free languages.

2 Definitions

Throughout this paper, we follow the terminologies and the notations as in [2], [9].

Definition 1 (Generalised splicing scheme) *Generalised splicing scheme is defined as a triplet $\sigma_G = (V_1, V_2, R)$, where V_1, V_2 are alphabets, and $R \subseteq V_1^* \# V_1^* \$ V_2^* \# V_2^*$. Here R can be infinite, and R is considered as a set of strings, hence a language. For a given σ_G , and a languages $L_1 \subseteq V_1^*$ and $L_2 \subseteq V_2^*$, we define*

$$\sigma_G(L_1, L_2) = \{z_1, z_2 : (x, y) \models_r (z_1, z_2), \text{ for } x \in L_1, y \in L_2, r \in R\}.$$

Given three families FL_1, FL_2, FL_3 ; we define

$$GS(FL_1, FL_2, FL_3) = \{\sigma_G(L_1, L_2) : L_1 \in FL_1, L_2 \in FL_2, R \in FL_3\},$$

i.e. $GS(FL_1, FL_2, FL_3)$ is the set of strings generated by splicing a language of FL_1 , and a language of FL_2 , by using a set of splicing rules in FL_3 .

Note 1. Whenever we refer ‘generalised splicing’, we mean generalised 2-splicing.

Definition 2 (Generalised self assembly) *Let $w_1 \in L_1, w_2 \in L_2$ be any two words. The generalised x -self assembly operation $GSA_x(w_1, w_2)$ over $(\varepsilon \neq) x \in \text{sub}(w_1) \cap \text{sub}(w_2)$ is defined as follows:*

$$GSA_x(w_1, w_2) = \{u_1xv_1, u_2xv_2, u_1xv_2, u_2xv_1 : w_1 = u_1xv_1, w_2 = u_2xv_2\}.$$

The self assembled words are the words that are generated when we trace from a left corner to a right corner in the figure 2. Given any two languages L_1 and L_2 , over the alphabet set V_1 and V_2 respectively, we define-

$$GSA(w_1, w_2) := \bigcup_x GSA_x(w_1, w_2),$$

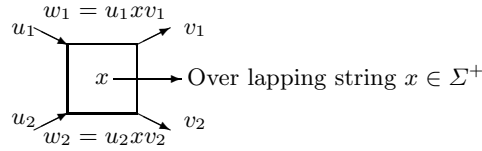


Fig. 2. Super impose over the common sub-string x

and

$$GSA(L_1, L_2) := \bigcup_{\substack{w_1 \in L_1 \\ w_2 \in L_2}} GSA(w_1, w_2).$$

Though a self assembly process will not include the parent words w_1 and w_2 (when $w_1 \neq w_2$), in the above definition, we purposefully include the parent words for the sake of more clarity of studying the GS through the GSA approach, i.e. we plan to investigate $GS(L_1, L_2, L_3)$ where $L_3 = V^+ \cup \{(w_1, w_2) : w_1 \in L_1, w_2 \in L_2\}$ (V is the set of common symbols that appear in L_1 and L_2). The pair (w_1, w_2) in the set of splicing rules means that w_1 will be cut after w_1 and w_2 will be cut after w_2 . Note that, the parent words w_1 and w_2 are included in $GS(w_1, w_2)$.

With the motivation given in section 1 and with the above two definitions, we have the following theorem.

Theorem 1. *gsags] Let L_1 and L_2 be any two languages. Let $V = V_{L_1} \cap V_{L_2}$, where V_{L_1} and V_{L_2} are the alphabets of L_1 and L_2 respectively. Then*

$$GS(L_1, L_2, R) = GSA(L_1, L_2),$$

where

$$R = V^+ \cup \{(w_1, w_2) : w_1 \in L_1, w_2 \in L_2\}$$

3 Generalised Self assembly of finite languages

This is the simplest and most trivial case. Suppose there are two finite languages L_1 and L_2 , each containing n_1 and n_2 words respectively. Given any two words, there can be only finitely many common symbols between them. So only finitely many new words can be generated by self assembly. Since the parent languages are finite the end product $S(L_1, L_2)$ contains only finite number of words. Thus we get the following theorem:-

Theorem 2. *Self assembly of two finite languages is finite. So we may write,*

$$GSA(FIN, FIN) = FIN.$$

4 Generalised Self assembly of regular languages

In this section we shall investigate behaviour of the self assembly of two regular languages. We know that regular languages can be generated by regular grammar and are also accepted by a finite automata. We shall show that self assembly of any two regular languages is regular. We shall prove it by both the automata and grammar approach.

4.1 Generalised Self assembly of regular grammar

In this section we shall describe: given any two regular grammars G_1, G_2 of languages L_1 and L_2 respectively, how to construct a grammar for the self assembly language $S(L_1, L_2)$.

Definition 3 (Self assembly of REG grammars) *Let $G_i = (N_i, T_i, R_i, S_i), i = 1, 2$, be the regular grammars of languages $L_1 = L(G_1)$ and $L_2 = L(G_2)$, where N_i 's are the set of non terminals, $N_1 \cap N_2 = \emptyset$, T_i 's are the set of terminals and $T_1 \cap T_2 \neq \emptyset$ (only then we can self assemble), S_i 's are the starting symbols and R_i 's are the rules respectively.*

The generalised self assembly of G_1 and G_2 , written as $GSA(G_1, G_2)$ is defined as

$$G = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, S, R), \quad S \notin N_1 \cup N_2,$$

where R includes the following rules:

1. $S \rightarrow S_1, S \rightarrow S_2$.
2. All the rules of R_1 and R_2 .
3. For $a \in T_1 \cap T_2$, for each pair of the rules $A \rightarrow aB \in R_1$ and $A' \rightarrow aB' \in R_2$, include the rules $A \rightarrow aB'$ and $A' \rightarrow aB$ in R .

Note 2. REG grammars are ones whose rules are of the form $A \rightarrow aB$ or $A \rightarrow a$, where A is non-terminal and a is a terminal. The two rules can be jointly expressed as $A \rightarrow a\gamma$ where γ is a non-terminal or $\gamma = \varepsilon$.

Example 1 Let $G_1 = (\{S_1\}, \{a, b\}, R_1 = \{S_1 \rightarrow aS_1, S_1 \rightarrow b\}, S_1)$, and $G_2 = (\{S_2\}, \{a, b\}, R_2 = \{S_2 \rightarrow bS_2, S_2 \rightarrow a\}, S_2)$. $L_1 = L(G_1) = a^*b$ and $L_2 = L(G_2) = b^*a$. Then the GSA grammar is $G = (\{S, S_1, S_2\}, \{a, b\}, R, S)$, where the rules R are given as

$$\begin{array}{ll} S \rightarrow S_1 & S_1 \rightarrow aS_1|b|bS_2|a \\ S \rightarrow S_2 & S_2 \rightarrow aS_1|bS_2|a|b. \end{array}$$

Note that the language generated by G , $L(G)$ will include the languages $L(G_1)$ and $L(G_2)$. Thus GSA of two regular grammars is again regular. In the same spirit of the above definition, we define GSA of linear grammars and GSA of context free grammars (for this, we consider the Greibach normal form for CFG).

Theorem 3. Let G_1 and G_2 be any two regular grammar. Then

$$L(GSA(G_1, G_2)) = GSA(L(G_1), L(G_2)).$$

Proof. Part I:

Case I $w \in L(G_1)$ or $w \in L(G_2)$. It is trivial, since the rules R_1 and R_2 are included in $GSA(G_1, G_2)$.

Case II $w \notin L(G_1)$ or $w \notin L(G_2)$. Let $w \in GSA(L(G_1), L(G_2))$. There exists $w_1 \in L(G_1)$, $w_2 \in L(G_2)$, $a \in \Sigma_{w_1} \cap \Sigma_{w_2}$, and $w = GSA(w_1, w_2) = uav$ such that $w_1 = uau_1$, $w_2 = v_1av$, where $u \in \text{prefix}(w_1)$, $v_1 \in \text{prefix}(w_2)$, $u_1 \in \text{suffix}(w_1)$, $v \in \text{suffix}(w_2)$.

Since $w_1 \in L(G_1)$, there exists a sentential form

$$S_1 \Rightarrow_{G_1}^* uA \Rightarrow uAB \Rightarrow_{G_1}^* uau_1 : A \rightarrow aB \in R_1$$

for deriving $w_1 = uau_1$. Similarly there exists a sentential form

$$S_2 \Rightarrow_{G_2}^* v_1A' \Rightarrow v_1AB' \Rightarrow_{G_2}^* v_1av : A' \rightarrow aB' \in R_2$$

for deriving $w_2 = v_1av$. Since $A \rightarrow aB \in R_1$ and $A' \rightarrow aB' \in R_2$ implies that $A \rightarrow aB' \in R(GSA(G_1, G_2))$, we have the sentential form

$$S \Rightarrow_{GSA(G_1, G_2)} S_1 \Rightarrow_{G_1}^* uA \Rightarrow_{GSA(G_1, G_2)} uAB' \Rightarrow_{G_2}^* uav'$$

i.e.

$$S \Rightarrow_{GSA(G_1, G_2)} uav = w.$$

Hence $w \in L(GSA(G_1, G_2)) \Rightarrow GSA(L(G_1), L(G_2)) \subseteq L(GSA(G_1, G_2))$.

Part II:

Let $w \in L(GSA(G_1, G_2))$. Without loss of generality, we assume that $w \notin L(G_1)$ and $L(G_2)$.

Since $w \in L(GSA(G_1, G_2))$, w can be expressed as $w = uav$. So there exists a sentential form

$$S \Rightarrow_{GSA(G_1, G_2)} S_1 \Rightarrow_{G_1}^* uA \Rightarrow_{GSA(G_1, G_2)} uAB' \Rightarrow_{G_2}^* uav$$

Since $A \rightarrow aB' \in R(GSA(G_1, G_2))$, but $\notin R_1, R_2$ (because $A, B \notin N_2$ and $A', B' \notin N_1$), there exists productions of the type $A \rightarrow aB \in R_1$ and $A' \rightarrow aB' \in R_2$.

This implies

$$S_1 \Rightarrow_{G_1}^* uA \Rightarrow_{G_1} uAB \Rightarrow_{G_1}^* uax, \quad \text{using the production } A \rightarrow aB$$

and

$$S_2 \Rightarrow_{G_2}^* yA' \Rightarrow_{G_2} yAB' \Rightarrow_{G_2}^* yav, \quad \text{using the production } A' \rightarrow aB'$$

This gives $\exists uax \in L(G_1)$ and $yav \in L(G_2)$ corresponding to $w = uav \in GSA(L(G_1), L(G_2))$.

Hence $L(GSA(G_1, G_2)) \subseteq GSA(L(G_1), L(G_2))$. Hence the result.

4.2 Generalised Self assembly of finite automata

If L_1 and L_2 any two *REG* languages, there exists two finite automatas M_1 and M_2 such that $L_1 = L(M_1)$ and $L_2 = L(M_2)$. While L_1 and L_2 can self assemble by string overlapping, it is interesting to explore whether the corresponding automata self assemble to an automata M such that the language of the self assembled automata is same as self assembly of languages. If a word w is accepted by a FA, every symbol a in w corresponds to an edge ' a ' in the transition diagram of the FA. This gives the idea that the FA's can be self assembled by the overlapping edge with the same level. Thus we have the following definition:

Definition 4 (Generalised self assembly of two FA's) Let $M_1 = (Q_1, V_1, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, V_2, \delta_2, q_2, F_2)$ be two machines such that $V_1 \cap V_2 \neq \emptyset$. The generalised self assembly of M_1 and M_2 written as $GSA(M_1, M_2)$ is defined as

$$M = (Q = Q_1 \cup Q_2, V_1 \cup V_2 \cup \{\varepsilon\}, \delta, q_0, F_1 \cup F_2).$$

δ is defined as follows

1. $\delta(q_0, \varepsilon) = \{q_1, q_2\}$.
2. $\forall a \in V_1 \cup V_2, q \in Q$

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \\ \delta_2(q, a) & q \in Q_2 \end{cases}$$

3. For every pair of transitions $\delta_1(q_i, a) = q_j$ and $\delta_1(q'_i, a) = q'_j$, $q_i \in Q_1$, $q'_i \in Q_2$, we include two new transition rules,

$$\delta(q_i, a) = q'_j \quad \delta(q'_i, a) = q_j.$$

Note that the language accepted by the GSA of M_1 and M_2 include $L(M_1)$ and $L(M_2)$.

It is observed that when G_1 and G_2 are regular grammars, we have

$$L(GSA(G_1, G_2)) = L(GSA(M_1, M_2)),$$

where $L(G_1) = L(M_1)$ and $L(G_2) = L(M_2)$.

The idea behind the self assembly of two FAs is the overlapping of the directed edge labelled with same symbol in the transition diagram of both the finite automatas. Every transition rules corresponds to a directed edge in the transition diagram. Let $\delta(q_i, a) = q_j$ and $\delta(q'_i, a) = q'_j$ be the transition in M_1 and M_2 respectively. In the self assembly of M_1 and M_2 , the directed edge in the transition diagram that corresponds to the above transition overlap: When the edges overlap, the states q_i and q'_i overlap. To add more clarity, the figure 3 is

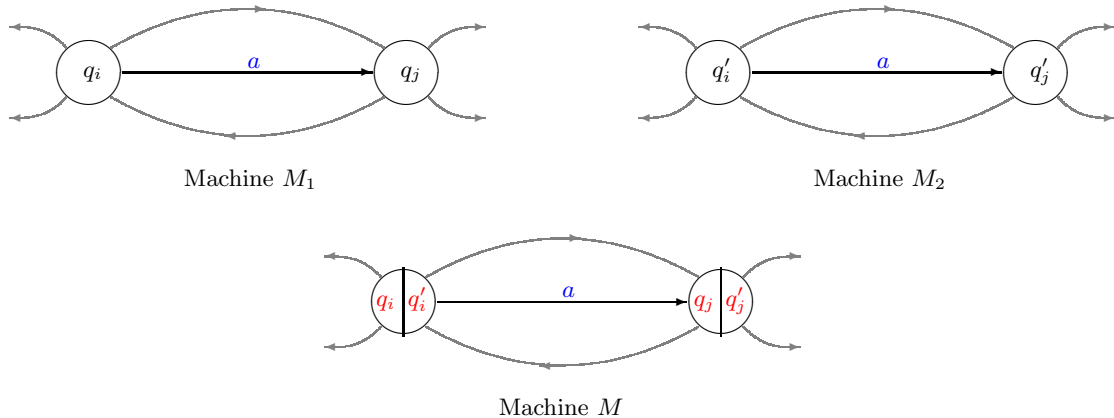


Fig. 3. Part of self assembled finite automata. Machine M_1 and M_2 are self assembled at the transitions edge a . The new FA M is drawn to specifically highlight the assembled states.

drawn in a way that all the transitions are preserved.

Theorem 4. *Let M_1 and M_2 be any two finite automatas. Then*

$$L(GSA(M_1, M_2)) = GSA(L(M_1), L(M_2)).$$

Proof. Without loss of generality, we assume that there is only one directed edge labeled a in the transition diagram of M_1 and M_2 , which can overlap. Further we can assume that all states of M_1 and M_2 are differently labeled.

Part I Let $w \in L(GSA(M_1, M_2))$.

Case I $w \in L(M_1)$ or $w \in L(M_2)$. Since $L(M_1), L(M_2) \subset GSA(L(M_1), L(M_2))$, we have $w \in GSA(L(M_1), L(M_2))$.

Case II $w \notin L(M_1)$ and $w \notin L(M_2)$. There exists a path from q_0 to any one of the final states involving the edge a the transition graph of M such that the path preceding the edge a is in M_1 (or in M_2), and the path succeeding the edge a is in M_2 (or in M_1).

$\Rightarrow w = w_1aw_2$, where $w_1\text{prefix}(x)$, $x \in L(M_1)$ (or $w_1\text{prefix}(x)$, $x \in L(M_2)$) and $w_2\text{suffix}(x)$, $x \in L(M_2)$ (or $w_2\text{suffix}(x)$, $x \in L(M_1)$); i.e. w_1 is the labels of the path in M_1 (or in M_2), and w_2 is the labels of the path in M_2 (or in M_1).

$\Rightarrow w$ can be written as the self assembly of the words $w_1aw'_1$ and w'_2aw_2 , where $w_1aw'_1 \in L(M_1)$ and $w'_2aw_2 \in L(M_2)$.

$\Rightarrow w \in GSA(L(M_1), L(M_2))$.

Hence $L(GSA(M_1, M_2)) \subset GSA(L(M_1), L(M_2))$.

Part II Let $w \in GSA(L(M_1), L(M_2))$.

$\Rightarrow w = GSA(x, y) : x \in L(M_1), y \in L(M_2)$.

$\Rightarrow w = w_1aw'_2$ or $w_2aw'_1$ where $x = w_1aw'_1$, $y = w_2aw'_2$.

\Rightarrow There exists a path with label w from q_0 to any one of the final states in the transition graph of M , involving the edge a .

$\Rightarrow w \in L(GSA(M_1, M_2))$.

Hence the result.

Combining the results above we get the following theorem.

Theorem 5. *Generalised self assembly of two regular languages is regular. So we may write,*

$$GSA(REG, REG) = REG.$$

We may also go a step further. For any $L_1 \in FIN$ we can generate an automata M_1 , in this way: for each word, make an automata which accepts only that word. All together this will make a finite automata, with a unique starting symbol, which may take the empty string ε and links to each of the individual automatas. Now given a regular language $L_2 \in REG$, we have an automata M_2 accepting it. We can self assembly them by the method described in theorem 4. The resultant is again a finite automata. Since $L_2 \subset GSA(L_1, L_2)$, by our construction, this automata also accepts infinite number of words. We can summarise this as:-

Theorem 6. *Self assembly of regular and finite languages is regular. So we may write,*

$$GSA(FIN, REG) = REG.$$

5 Generalised Self assembly of linear languages

Linear languages (written as LIN) are the ones which are characterised by the following grammar rules.

$$X \longrightarrow P_1YP_2 \qquad X \longrightarrow P, \qquad (1)$$

where X and Y are non-terminals (N), and P_1, P_2, P_3 are words over terminals(T) [2]. If P_1 (resp. P_2) is ε the grammar is called left-linear (resp. right-linear). Any linear language can be generated by right (or left) linear grammar. Also they are equivalent [2]. Hence for our purpose we convert all the grammars of the form of right-linear only, i.e. we are only considering rules of the form:

$$X \longrightarrow P_1Y \qquad X \longrightarrow P,$$

where $P_1, P_2 \in T^+$. Again we may further introduce new non-terminals, such that each rule is of either of the form:

$$X \longrightarrow aY \qquad X \longrightarrow a, \qquad (2)$$

where $a \in T \cup \{\varepsilon\}$ and $Y \in N^+$.

Method for self assembly of LIN grammar:

Now we use similar process as given in definition 3. Suppose we have $L_1, L_2 \in LIN$. We construct grammar $G_i = (N_i, T_i, S_i, R_i)$, $i = 1, 2$ for them such that $N_1 \cap N_2 = \emptyset$ and R_i 's are of the form of equation 2.

Define a grammar $G = (N, T, S, R)$ where $N = N_1 \cup N_2$, $T = T_1 \cup T_2$, S is the new starting symbol, and the rules of R are:

1. $S \longrightarrow S_1, S \longrightarrow S_2$.
2. All the rules of R_1 and R_2 .
3. For $a \in T_1 \cap T_2$, for each pair of the rules $A_1 \longrightarrow a\gamma_1 \in R_1$ and $A_2 \longrightarrow a\gamma_2 \in R_2$, include the rules $A_1 \longrightarrow a\gamma_2$ and $A_2 \longrightarrow a\gamma_1$ in R , where $\gamma_1 \in N_1^*$ and $\gamma_2 \in N_2^*$.

The analogous result of theorem 3 follows the same line of argument. Thus we can also conclude that:

Theorem 7. *Self assembly of two linear languages is linear; i.e.*

$$GSA(LIN, LIN) = LIN.$$

6 Generalised Self assembly of context free languages

We self assemble CF grammars, and thus show that the self assembly of two CF languages is again a CF language. Instead of using general grammar rules, we take the help of Greibach normal form [6]. To use this, we can assume without loss of generality, that the parent languages are ε free. Now, in Greibach normal form each rule is of the form $A \longrightarrow a\gamma$, where $\gamma \in N^*$. We use exactly the same method used for linear grammar. Same lines of arguments give us:

Theorem 8. *Generalised self assembly of two context free languages is context free; i.e.*

$$GSA(CF, CF) = CF.$$

7 Conclusion

In all definitions of GSAs of languages, grammars (definition 3) and FAs (definition 4), the parent words are included in the words generated by the GSA. In fact, in any self assembly process of w_1 and w_2 , $w_1 w_2$ will be generated only when $w_1 = w_2$. But, in our definition of GSA, we prefer to include w_1 and w_2 (even if $w_1 \neq w_2$) in $GSA(w_1, w_2)$ with a purpose. Though we can define the GSA of grammars (as well as FAs) so that the parent words are not included in the words generated, the process will be highly complicated. The main purpose of this paper is just to study the generalised splicing in the self assembly approach. For the sake of not loosing clarity of our approach in this study, we prefer to include the parent words in all our definitions, namely GS of languages, GSA of languages, and GSA of grammars.

Thus, we have proved that $GS(FIN, FIN, R) = FIN$, $GS(REG, REG, R) = REG$, $GS(FIN, REG, R) = REG$, $GS(LIN, LIN, R) = LIN$ and $GS(CF, CF, R) = CF$, where R is as mentioned as in Theorem 1. This study can further be extended to study the other generalised splicing classes of languages.

References

1. L. Adleman, Towards a mathematical theory of self-assembly, *Technical Report (00-72)*, University of South California, 2000.
2. Arto Saloma, Formal Languages, Academic Press Inc. 1973.
3. Karel Culik II, Tero Harju, Splicing semigroups of dominoes and DNA, *Discrete Applied Mathematics*, **31 (3)**, 261-277, 1991.
4. Erzsébet Csuhaj-Varjú, Ion Petre, György Vaszil, Self assembly of strings and languages, *Theoretical Computer Science*, **374 (1-3)**, 74-81, 2007.

5. Tom Head, Formal language theory and DNA : An analysis of the generative capacity of specific recombinant behaviours, *Bull. Math. Biology*, **49**, 737-759, 1987.
6. John Hopcroft, Rajeev Motwani, Jeffrey Ullman, Introduction to automata theory, languages, and computation (2e), Pearson Indian reprint, 2001.
7. Gh. Păun, On the Splicing operation, *Discrete Applied Mathematics*, **70**, 57-79, 1996.
8. Gh.Păun, Grzegorz Rozenberg, Arto Salomaa, Computing by Splicing, *Theoretical Computer Science*, **168(2)**, 321-336, 1996.
9. Gh. Păun, Grzegorz Rozenberg, Arto Salomaa, DNA Computing : New Computing Paradigms, Springer-Verlag, 1998.